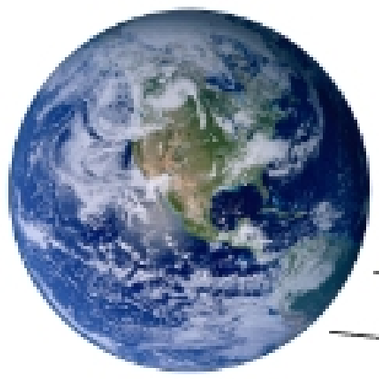
CMSC 631 – Program Analysis and Understanding
Fall 2003

Abstract Interpretation

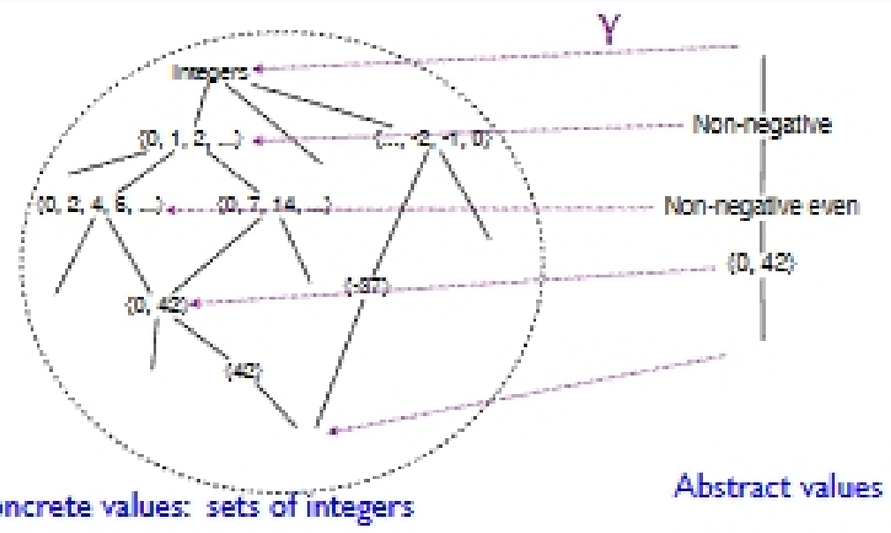
Based on lectures by David Schmidt and by Alex Aiken

What is an Abstraction?

- A property from some domain

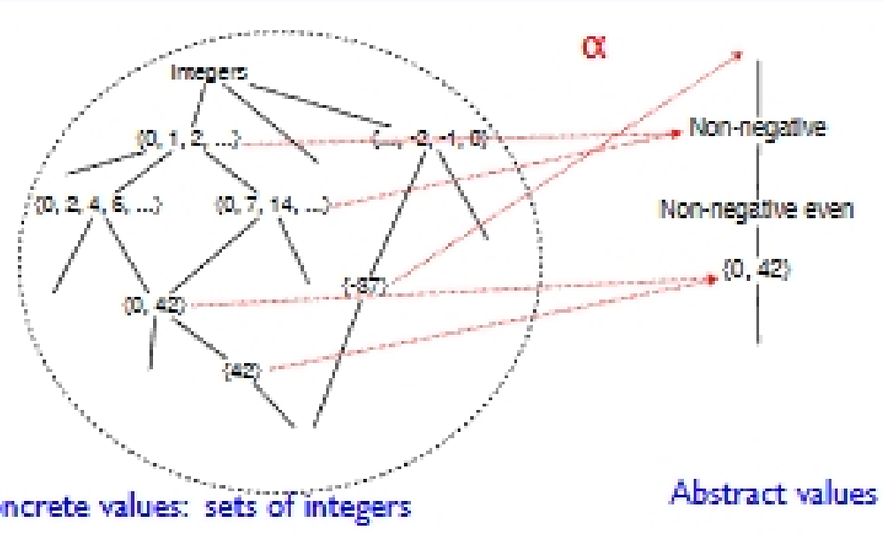


Example Abstraction



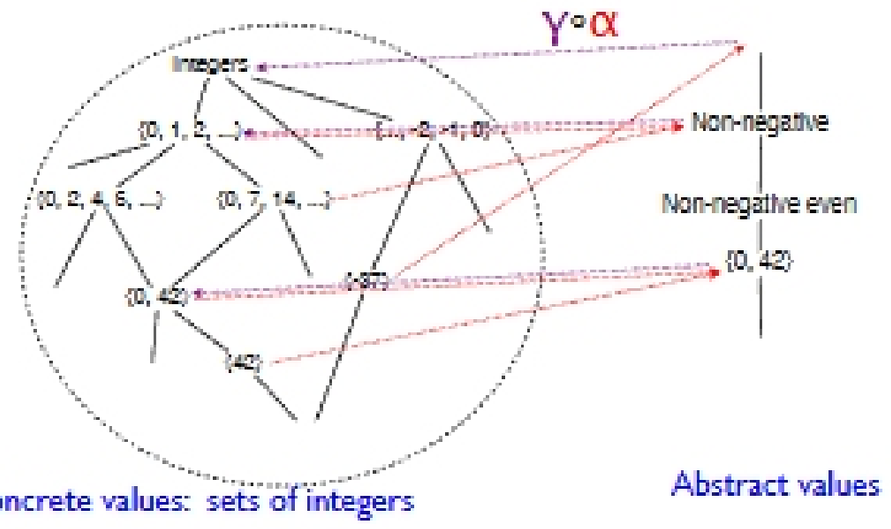
Concretization function γ maps each abstract value to concrete values it represents

Abstraction is Imprecise



Abstraction function α maps each concrete set to the best abstract value

Composing α and γ



Abstraction followed by concretization is sound but imprecise

α and γ Form a Galois Insertion

- α and γ are monotonic
  - Recall: f is monotonic if $x \sqsubseteq y$ $f(x) \sqsubseteq f(y)$
  - Also called "order preserving"
- S $\gamma(\alpha(S))$ for any concrete set S
- $\alpha(\gamma(A))$ A for any abstract element A

- Next up: Abstract interpretation in action
  - We'll develop an abstract interpretation of a simple language and prove it correct using these ideas

Source Language

- Integers and multiplication
  - e ::= i | e * e
- Standard semantics of the program
  - Eval : e → Int
  - Eval(i) = i
  - Eval(e1*e2) = Eval(e1) × Eval(e2)

Abstraction

- Define an abstract semantics that computes only the sign of the result

$\hat{\times}$	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

$AEval : e \rightarrow \{-, 0, +\}$

$$AEval(i) = \begin{cases} + & i > 0 \\ 0 & i = 0 \\ - & i < 0 \end{cases}$$

$AEval(e1*e2) = AEval(e1) \hat{\times} AEval(e2)$

Soundness

- We can show our abstraction correctly predicts the sign of an expression
- Proof: by structural induction on e
  - Eval(e) > 0 iff AEval(e) = +
  - Eval(e) = 0 iff AEval(e) = 0
  - Eval(e) < 0 iff AEval(e) = -

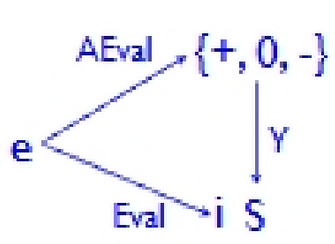
Another Approach to Soundness

- Natural concretization function

$$\gamma(+) = \{i \mid i>0\}$$
$$\gamma(0) = \{0\}$$
$$\gamma(-) = \{i \mid i<0\}$$

- Note: This presentation is slightly non-standard
  - Usually defined in terms of execution traces

Soundness (cont'd)



- Our abstraction is sound if
  - Eval(e) $\gamma(AEval(e))$
- Soundness proof: later

Adding Unary Negation

- e ::= i | e * e | -e
-
- Eval(-e) = -Eval(e)
- $AEval(e) = \hat{-}AEval(e)$
-
-

$\hat{=}$	+	0	-
	-	0	+

-
- No problems

Adding Addition

- e ::= i | e * e | -e | e + e
-
- Eval(e1+e2) = Eval(e1) + Eval(e2)
- AEval(e1+e2) = AEval(e1) $\pm$ AEval(e2)

$\pm$	+	0	-
+	+	+	?
0	+	0	-
-	?	-	-

Our abstract domain is not closed under addition

Solution

- Add an abstract value to represent any integer
- Finding closed domain often key design problem

$\gamma(\top)$ = {integers}

$\pm$	+	0	-	$\top$
+	+	+	$\top$	$\top$
0	+	0	-	$\top$
-	$\top$	-	-	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

- Other operations also need to handle $\top$

Two Ways to Lose Information

- OK: Abstraction still precise enough
  - Eval((5 * 5) + 6) = 31
  - AEval((5*5) + 6) = (+ $\pm$ +) $\pm$ + = +
    - Abstractly, we don't know which value we computed
    - ...but we don't care, since we only want the sign
- Not so good: "Don't know" values
  - Eval((1 + 2) + -3) = 0
  - AEval((1 + 2) + -3) = (+ $\pm$ +) $\pm$ - = + $\pm$ - = $\top$
    - We also don't know which value we computed
    - ...and we can't even figure out its sign

Adding Integer Division

- What happens when we divide by zero?
  - The result is not an integer
  - If we divide each integer in a set by 0, the result is the empty set

$\gamma(\ ) = \varnothing$

$\div$	+	0	-	$\top$
+	+	0		$\top$
0				
-		-	+	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

Adding Integer Division (cont'd)

- We need to extend other abstract operations to work on
- Every operation involving results in
  - All operations are *strict* in

$$\_ \ a = \_$$
$$a \ \_ = \_$$
$$\pm \ a = \pm$$
$$a \ \pm = \pm$$
$$\div = \div$$

The Abstract Domain

- Look, ma, a lattice!
- We've got:
  - A set of elements { +, 0, -, $\top$}
  - A relation $\leq$ that is
    - Reflexive
    - Anti-symmetric
    - Transitive
  - And
    - The least upper bound (lub, $\sqcup$) and greatest lower bound (glb, $\sqcap$) exists for any pair of elements
    - So it's a lattice